HW4: Generative Models - VAE and GAN

Srivatsan Srinivasan
srivatsansrinivasan@g.harvard.edu

Sebastien Baur
sebastienbaur@g.harvard.edu

Bernd Huber
bhb@seas.harvard.edu

March 30, 2018

Github Repo : `https://github.com/harvard-ml-courses/cs287-s18-sb_ss`

1 Introduction

In this assignment, we are tackling the problem of generative models for images. The task is to learn a model that can synthesize from a latent variable distribution having a simple prior. Depending on the model we use, the posterior can be either implicit (impossibility to perform inference), or we can have a way to perform inference via an encoder network that approximates the true posterior. To construct deep generative models, we use two widely used methods to train these latent space models - Variational Autoencoders and Adversarially trained generative models. For the purposes if this homework, we will be using the binary (for VAE) or continuous (for GAN) handwritten digits dataset - MNIST. The next section will introduce key notations and explain the problem under the lens of VAEs and GANs, followed by model description, variants and architectures followed by a section that qualitatively and quantitatively discusses results.

2 Problem

We are going to use these notations in the problem description and beyond and any deviance will be noted in the corresponding section.

- $\boldsymbol{x}, p(\boldsymbol{x})$: True data and probability distribution generating the true data
- $\boldsymbol{y}$: Auxiliary inputs to data such as constraints/class labels.
- $\boldsymbol{z}, p(\boldsymbol{z})$: Latent Variables that are considered to generate the data and the probability distribution in the latent space.(sometimes used to denote prior on latent variable)
- FC, Conv, RNN : Fully connected layer, Convolutional Layers, Recurrent Neural Networks(Variants of GRU/LSTM etc.)
- G, D : Generator and Discriminator networks in GAN.

- KL, JS, EM : KL Divergence, Jenson Shannon Divergence, Earth Movers Distance

Under these models, a generative model could be thought of as learning a function approximation to the distribution $p(\boldsymbol{x})$. Latent Variable models, characterized by the latent variables given by $p(\boldsymbol{z})$, assume that the latent distribution generates the data and the joint data generation distribution could now be learned as $p(\boldsymbol{x}, \boldsymbol{z}) = p_\theta(\boldsymbol{x} \mid \boldsymbol{z})p(\boldsymbol{z}) = p_\theta(\boldsymbol{z} \mid \boldsymbol{x})p(\boldsymbol{x})$. Variational models introduce a new family of variational parameters q_λ to approximate the conditional distribution when the posterior becomes computationally intractable and sets up an approximate bound to the inference from the true distribution. GANs on the other hand learn the $p_\phi(\boldsymbol{x} \mid \boldsymbol{z})$ directly through a network parametrized by ϕ. Another variant we were interested is the conditional generation problem which could be thought of as modeling the distribution $p(\boldsymbol{x}, \boldsymbol{z} \mid \boldsymbol{y})$ given the conditional variables.

3 Models

3.1 Variational Autoencoders

3.1.1 General Theory

Variational autoencoders (Kingma and Welling (2013)) allows to perform variational inference using deep neural networks. They postulate the existence of usually low dimensional latent variables that explain the data. In the case of the MNIST dataset, these latent variables can be, for example, qualtiative aspects of digit that is written (0,1,2,3...), or the style of writing (how smooth, the angle,...). A special class of variational inference models called variational autoencoders are trained to reconstruct its input, and it has an additional distributional constraint on the latent space, forcing it to decode the whole of it into valid data samples. The objective is therefore the classical ELBO:

$$\underset{\theta}{\text{maximize}}\ \mathbb{E}_{z\sim q_\theta(\cdot|x)} \log p_\theta(x|z) - KL(q_\theta(\cdot|x)||p(\cdot)) \leq \log p(x)$$

Where $q_\theta(\cdot|x)$ approximates the true posterior (it is a diagonal Gaussian whose parameters are parametrized by a neural network). The prior on z is a standard Gaussian.

For a Conditional VAE(CVAE) (Sohn et al. (2015)), we have instead:

$$\underset{\theta}{\text{maximize}}\ \mathbb{E}_{z\sim q_\theta(\cdot|x,c)} \log p_\theta(x|z,c) - KL(q_\theta(\cdot|x,c)||p(\cdot)) \leq \log p(x)$$

For PixelCNN (van den Oord et al. (2016)) (Gulrajani et al. (2016)), the decoder is autoregressive using causal convolutions. In practice, it means that the decoder has another input, which is the picture it is trying to predict. The convolutions are shifted so that the output depends only on the previous pixels. It is trained via teacher forcing (the predicted pixels depend only on the "previous" pixels of the true image). At testing, the model generates a picture using its own outputs as inputs, starting from a zero image. The generation is sequential and therefore takes much more time than when the pixels in the output were independently generated.

3.1.2 Model variants, architectures and hyperparameters

Note that for all models, we used batch normalization (Ioffe and Szegedy (2015)) and ReLU or LeakyReLU activation. The architectures are kept simple and we didn't try to experiment with the impact of hyper parameters on the performance of the model. Optimizer is Adam (Kingma and Ba (2014)) with learning rate 10^{-2}.

1. **Vanilla VAE (VAE)** Both the encoder and the decoder are simple feed forward neural networks (one hidden layer). We used small models (100 hidden dimension, ReLU, BatchNorm, latent dim 2) as they were fast to train and the latent space is easy to visualize

2. **Vanilla Convolutional VAE** This time, the encoder is a CNN (2 layers, kernel size 3, stride 1, padding 1). The architecture of the decoder didn't change.

3. **Vanilla conditional VAE** Both the encoder and the decoder are set to be feed forward neural network (same architecture as 1.). Both of them have an additional input, which is the label of the image. It allows to perform conditional generation, i.e it is possible to generate variants of a given number. In that case the latent space encodes the "style" of the generated digit.

4. **PixelVAE** This time, the decoder is a PixelCNN. It is an autoregressive model. It is supposed to achieve much better results. Intuitively, the values of the pixels depend on the neighboring ones: two adjacent pixels are likely to take similar values. This model is able to exploit this and is supposed to remove the blurriness of generated pictures. **We implemented a pixelCNN based VAE but our training was not satisfactory and hence we don't present its results.**

3.2 Generative Adversarial Networks - General Theory

Adversarial generative networks were first introduced in 2014 by Goodfellow et al. (2014). GANs can be thought of as a two player minimax game between a generator network and a critic alias discriminator network. In the most generic form of GAN, the critic's role is to assess the quality of synthetic sentences and demarcate(with confidence) the synthetic corpus of sentences from real corpus. Asymptotically, when the generator produces sentences of the same nature as the real corpus, critic has a 50% classification confidence.The generator's objective is to ensure that the discriminator's confidence (the probability it assigns a sentence as belonging to real data vs synthetic data) is made as high as possible. Thus, we see that both networks are optimizing for two conflicting objectives and we can train them simultaneously using a minimax framework. Let us introduce certain formal notations we use to define a GAN. Let us call G, D as the generator and the discriminator networks respectively where the generator outputs a data sample and the discriminator outputs a probability score. p_s, p_t represent the synthetic data and true data respectively. Then the minimax objective of the game could be formalized as:

$$V(G, D) = \mathbb{E}_{p_t} \log D(x) + \mathbb{E}_{p_s} \log(1 - D(x))$$

On the other hand, the generative model is shown on Figure 1. $P(\boldsymbol{x} \mid \boldsymbol{z})$ is parametrized by a neural network $G(\phi)$ The training of GANs thus follows multiple iterations of this sequence of steps, often training discriminator and generator with different schedules. Here we present a simple version where each one is trained at every step.

Figure 1: Graphical model structure of GANs (also true for the assumptions made in VAE models). Every observed variable is influenced by every latent variable.

1. Sample batch of n latent variables $z_1....z_n$ from the generator's prior $p_g(z)$
2. Sample batch of n real data from the data $x_1....x_n$ generating distribution
3. Perform gradient update on the discriminant objective

$$V(D;\theta) = \sum_{i=1}^{n} \log(D(x_i)) + \log(1 - D(G(z_i)))$$

4. Sample batch of m latent variables $z_1....z_n$ from the generator;s prior.
5. Perform gradient updates on the generator objective.

$$V(G;\phi) = \sum_{i=1}^{n} \log(1 - D(G(z_i)))$$

The advantages of GAN over other famous generative frameworks include

- Very little architectural restrictions on generator
- GANs are proven to be asymptotically consistent and no variational bounds are needed
- Does not make any Markov chain assumptions(like boltzmann machines) and can generate samples in parallel.

3.2.1 Model Variants, Architectures and Hyperparameters

We tried different variants to the GAN architectures and we explain the reasoning behind these incremental choices and the set of hyperparameters used in this section.
Note that we didn't use batch normalization for the discriminator because it receives data froom two sources having different distribution, one being non-stationary. Therefore the mean and the variance of each activation layers have no reason to converge, and it just adds instability to training.

1. **Vanilla GAN(GAN)** - Vanilla GAN consists of a generator and discriminator that are both simple feed-forward neural networks. The training procedure involved training generator and discriminator for 100/200 epochs. The architecture that we used in this assignment can be seen follows. Let **Z** - 100 dimensional latent variable with standard gaussian prior.

   GENERATOR

$\boldsymbol{Z} \rightarrow FC(256) \rightarrow FC(512) \rightarrow FC(1024) \rightarrow FC(28*28) \rightarrow \boldsymbol{X}$

DISCRIMINATOR

$\mathbf{X}(28*28) \rightarrow FC(1024) \rightarrow FC(512) \rightarrow FC(256) \rightarrow FC(1)$ [Sigmoid]

Apart from this, we used Adam optimizer, learning rate of 0.0002, 0.3 dropout and LeakyReLU activation functions.

2. **Conditional Vanilla GAN(CGAN)** - The next model is a direct extension of the first, in that we want to condition both the generator and discriminator(Mirza and Osindero (2014)). In our case, we use the digits' class labels as a multi-dimensional one-hot vector $\mathbf{y}$. The intuition behind this idea is that conditioning on the class labels allows the model to learn multi-modal distributions better as the class label helps the generator network to explore multiple modes of the distribution more extensively.

   GENERATOR

   $[\boldsymbol{Z} \mid \boldsymbol{y}] \rightarrow FC(256) \rightarrow FC(512) \rightarrow FC(1024) \rightarrow FC(28*28) \rightarrow \boldsymbol{X}$

   DISCRIMINATOR

   $[\mathbf{X}(28*28) \mid \boldsymbol{y}] \rightarrow FC(1024) \rightarrow FC(512) \rightarrow FC(256) \rightarrow FC(1)$ [Sigmoid]

   Apart from this, we used Adam optimizer, learning rate of 0.0002, 0.3 dropout, and Leaky ReLU activation functions.

3. **De-convolutional GAN(DCGAN)** - The success of convolutional networks in image recognition has been pretty well documented in computer vision. But, adapting a CNN to image GANs has been not so straightforward, as a simple adoption of a convolutional net followed by FC layers to GAN produces poor results with serious issues such as mode collapse and instability of the learned model(Radford and Metz (2016)). To allay these concerns, the architecture that we employed is very similar to the successful work of citeDCGAN which is underpinned by three key model decisions. The entire model decision is illustrated in Figure 2

   - Replace spatial pooling functions with strided convolutions so that the network learns its own spatial down-sampling
   - Eliminate FC layers on top of Convolutional layers similar to the state-of-the-art image classification models
   - Employ batch-normalization to stabilize learning and prevent mode-collapses through standardization of input to each layer.

4. **Conditional Deconvolutional GAN(CDCGAN)** - The conditional DCGAN is an exact analogy to DCGAN as CGAN was to GAN. We condition both the generator and discriminator on the class labels $\boldsymbol{y}$. The input to both the generator and discriminator is concatenated with $\boldsymbol{y}$ while following a similar architecture and training parameter(as DCGAN) on the rest of the neural network.

5. **Wasserstein GAN** - Wasserstein GANs was introduced in 2017(Arjovsky et al. (2017)) and has been suggested to improve the stability of GAN learning, get rid of problems such as mode collapse and provide meaningful learning curves. WGANs implicitly try to minimize the EM(Earth Movers) distance between the synthetic and real distribution. It enforces a

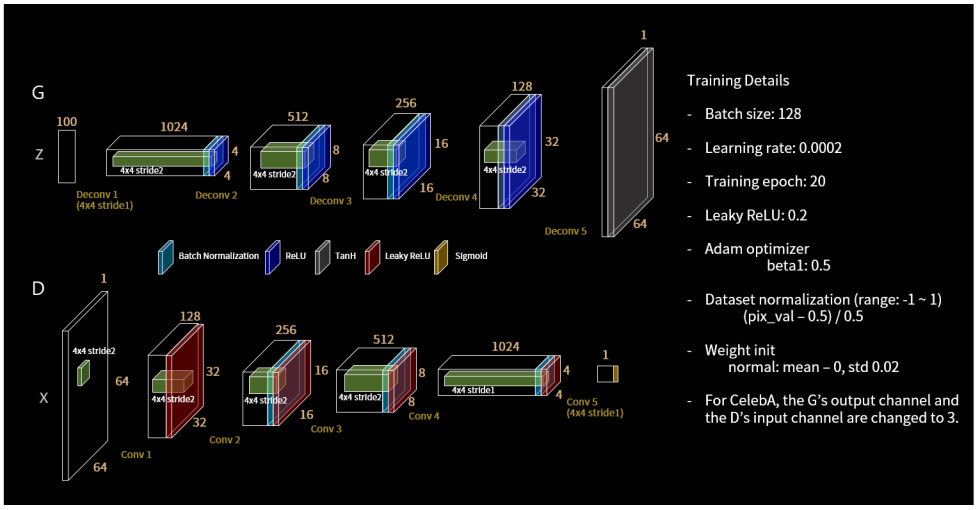


Figure 2: DCGAN architecture. Source:Radford and Metz (2016)

K-Lipschitz constraint on the functional form of the NN which in turn could be realized if the weights are curtailed to a finite hyper-cube say $W \in [-0.01, 0.01]^n$. The subtle changes between GAN and WGAN that were part of our implementation include :

- Remove the sigmoid that is applied on the discriminator output and allow it to be unbounded.
- Clip the weights to finite bounds after each gradient descent update.

Why are WGANs expected to yield more stable training than GAN(Arjovsky et al. (2017))?

(a) Minimizing the objective of traditional GANs, when the discriminator is optimal, is equivalent to minimizing the JS divergence between the true data distribution and the synthetic distribution. Contrary to the Wasserstein distance, it is not continuous, meaning that when the parameter approaches the optimal parameter, the distance is not necessarily going to 0. It makes the loss an unreliable proxy to evaluate whether the model is doing well or not.

(b) Another problem of traditional GANs is that the output of the discriminator is a sigmoid that can saturate. It means that if you train the discriminator to optimality (which is necessary to obtain reliable gradients), then the generator doesn't receive significant training signal because the gradients vanish. WGAN has been shown to greatly alleviate this issue, because it cannot saturate since the output of the discriminator is unbounded.

(c) A last and important problem of traditional GANs is their tendency to collapse to a few modes of the training data. Basically, they find realistic looking images (the modes of the discriminator distributions), and output only these, resulting in a very poor sample diversity. This is also due to the JS divergence minimization. When the discriminator is optimal, all its mass is placed on a few modes, and the only way (because of discontinuity of the JS divergence) for the generator to achieve good performance is to match

exactly these modes. The Wasserstein objective does not suffer this problem because its loss is continuous with respect to the parameter.

4 Results and Qualitative Discussion

Unlike previous homeworks, we were less concerned with hyperparameter search this time as we rather concentrated on implementing different variations in model architectures and training procedures in order to understand intrinsic properties of these models and how they connect to the generation problem we are trying to solve. Having said that, we are fully aware of the pronounced impact of hyperparameter choice on both the VAE and GAN and state that our decision was motivated largely by our exploratory interests than fine-tuning model performance.

4.1 Variational AutoEncoders Results

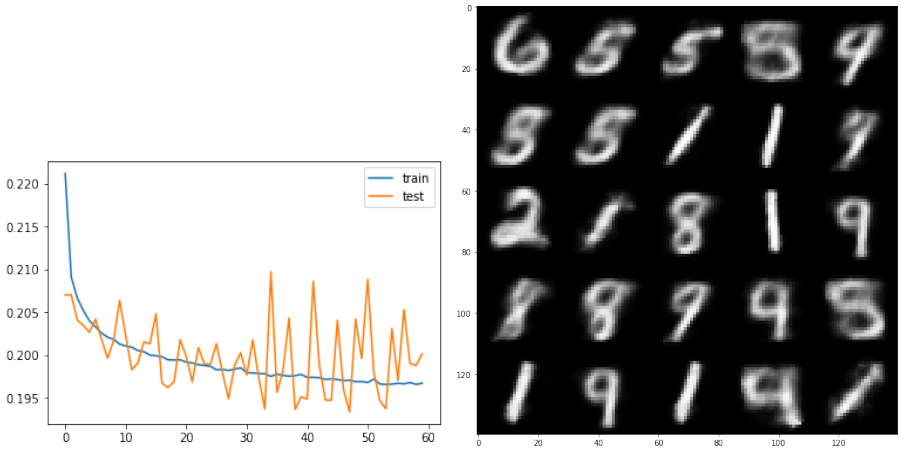
4.1.1 Remarks

1. **Vanilla vs CNN VAE -** We expected to see a significant performance difference between using convolutions vs simple feedforward networks in our neural nets as we thought that convolutions allow to capture local information. This in turn should help the discriminator to better tell the difference between true and synthetic samples, and therefore should provide a better learning signal. We were surprised to note that the generated samples for the vanilla VAE and the convolutional VAE were not significantly different and utilizing CNN architectures did not radically alter model performance in our case.

2. **BATCH NORMALIZATION -** Batch normalization helps in standardization of inputs. When training VAE, batch normalization tremendously expedited convergence: the validation loss of a BNVAE reaches the converged value of vanilla VAE in 2 epochs (instead of 15).

3. **CONDITIONING -** Conditioning is another factor that helped improve the quality of images generated. Conditioning the VAE(CVAE) on class labels implicitly gave some information to decoder on which parts of the image would most likely be brighter for a given class.

4.1.2 VAE - Generated Images and Discussion



(a) Training (b) Generation

Figure 3: Losses over training the VAE (a), and generated samples (b). Note that (b) actually does not represent the samples themselves, but rather the joint distribution on the pixels. Contrary to GAN samples, the pixels tend to be blurry (the probabilities are not very close to either 0 or 1), which is probably a consequence of our modeling assumptions. We assumed that the pixels are independent from each other. The decoder therefore uses no information about neighboring pixels, which tend to give a lot of information about the actual value of the considered pixel (if all your neighbors are black you are probably black, too). It is therefore hard to generate bright pictures.

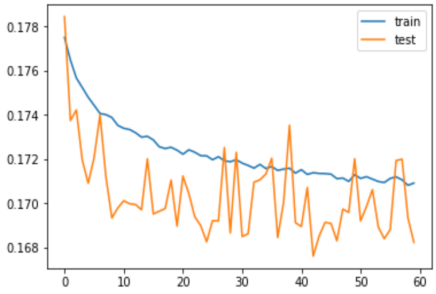
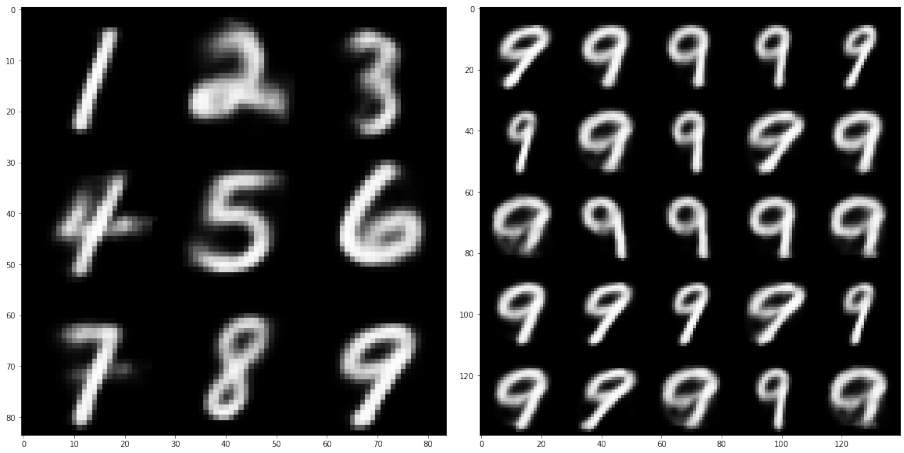


Figure 4: Training loss of the CVAE - Note that it achieves lower loss. It is because using an additional input (the digit) to the decoder gives it some prior information about the localization of bright pixels. For example, most 1s tend to have bright pixels in the middle vertical line. We can hope for brighter samples



(a) Choosing digit (b) Fixed digit, varying latent

Figure 5: CVAE does better - One can immediately see that CVAE samples are brighter, as was explained in 4. In (a), you can see that it allows for controlled generation, which is very interesting in itself. Generating random things is fascinating but not very useful. The truly interesting thing is the ability to condition the generation. On (b), you can see the impact of the latent code on the generation. The digit stays a 9, but the latent variations allows to generate different versions of it.

4.1.3 Latent space exploration

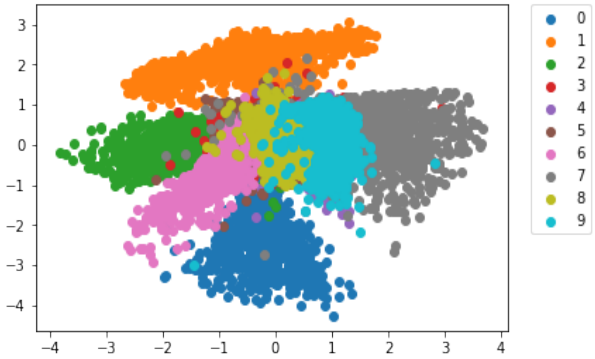
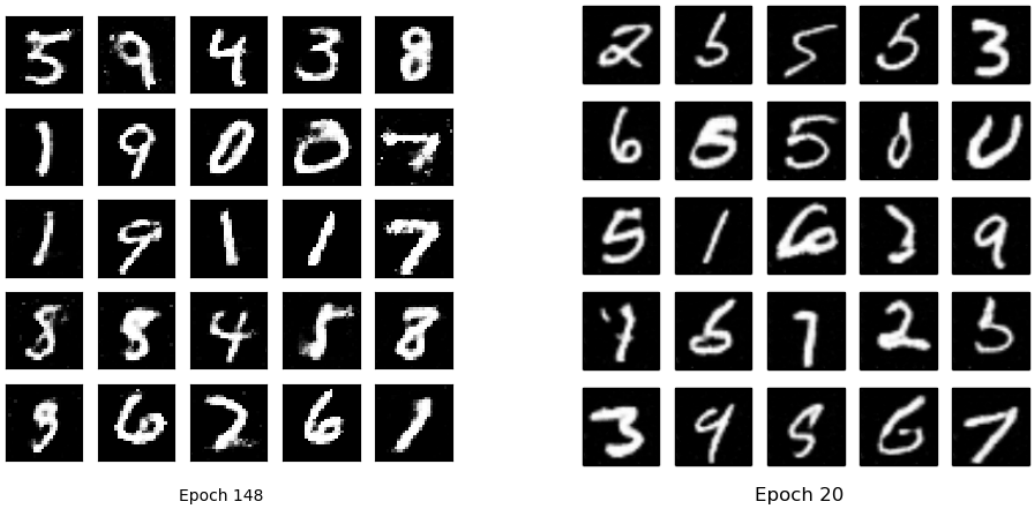


Figure 6: When you train a VAE, the hope is that the latent representation it learns is interpretable. You posit the existence of latent variables that would explain the data, however the model is never shown any label that would tell him what these high-level features are. And yet, as shown in this picture, it naturally clusters the latent representations of a given digit together. Besides, you can see that digits that can be similar depending on the writing style share common frontiers. For example, the lower part of 0s and 6s tend to be similar, so the dark blue and pink areas are close together; the upper and the right parts of 9s and 7s tend to be similar, so the grey and cyan areas share a frontier.

4.2 Results and Remarks on GAN training

4.2.1 GAN,DCGAN and Conditioning

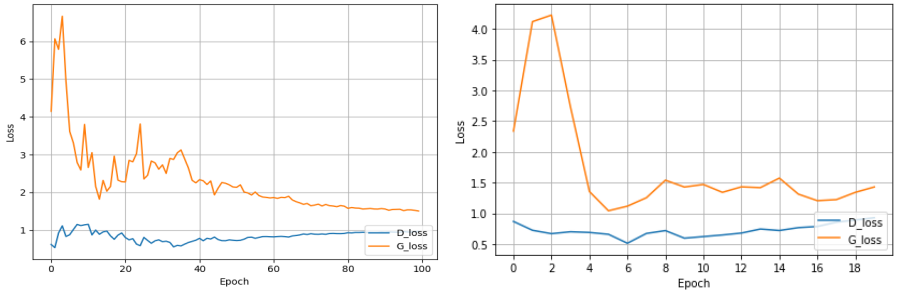
- **GENERATED IMAGES -** Figure 7 shows the images generated by a GAN and DCGAN without any conditioning. We can see that the images produced by a DCGAN are much sharper, but on the other hand, it took much longer to train because the network architecture involves deep convolutional layers We note that DCGAN was able to generate images of similar or better quality than conventional GANs at 150 epochs by around epoch 20.



(a) Vanilla GAN(GAN) (b) Deconvolutional CGAN(DCGAN)

Figure 7: Generated GAN(no conditioning) images

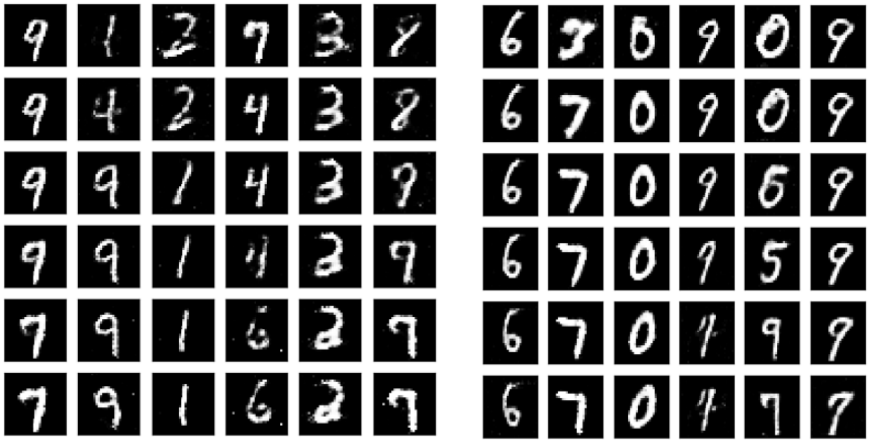
- **LOSSES -** Figure 8 shows the loss plots in training GAN and DCGAN. We see that the loss incurred by the generator is initially pretty high and it reduces as the number of training epochs increases.



(a) GAN (b) DCGAN

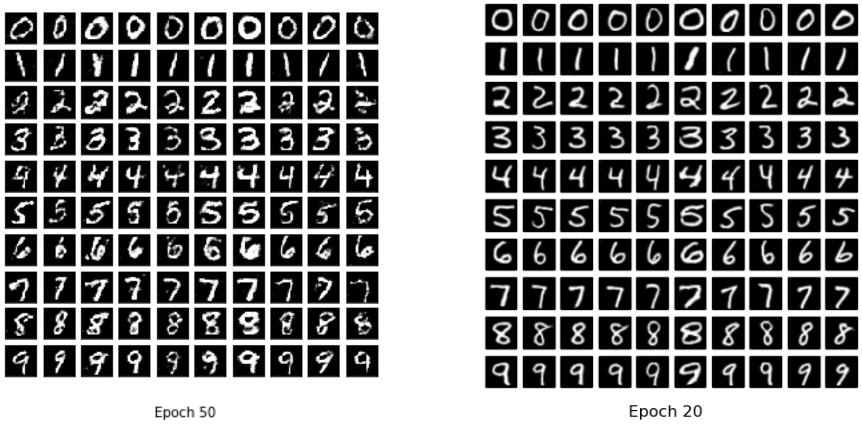
Figure 8: Loss plots for GANs(No conditioning)

- **LATENT VARIABLE INTERPOLATION -** Figure 9 shows some examples from both GAN and DCGAN in interpolation of latent vectors. We could see from certain examples(top to bottom) how digits metamorphose from one latent vector to another when we slowly blend them in steps of 0.2. We realize the advantages of learning a continuous latent variable during GAN training, which allows to generate images that are mixtures of two images or generate images with subtle differences from the base image.



(a) GAN (b) DCGAN

Figure 9: Interpolated Results - Six examples(6 z pairs) from left to right. z_1, Z_2 are aribitrarily sampled latent vectors. Top to bottom indicate progression of latent vector from z_1 to z_2 in steps of 0.2



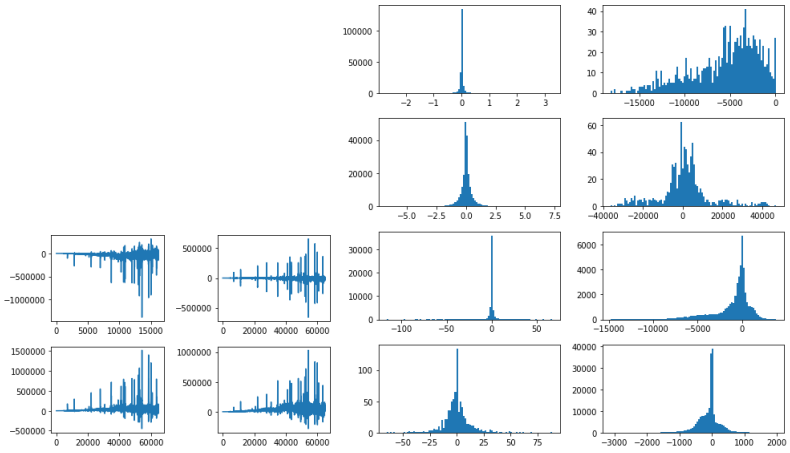
(a) Conditional Vanilla GAN(CGAN) (b) Conditional DCGAN(CDCGAN)

Figure 10: Generated Conditonal GAN images

As a recurrent theme across training these GAN models, we realized the following observations

- The training was unstable. Many times, with different architecture and hyperparameter choices, we did not see a smooth pattern of convergence in the loss plots. Losses did not seem a great proxy for the quality of digits generated(due to stagnation or sometimes even explosion) and we could verify sample quality only through visual inspection.
- Also,with certain trials, we also observed some form of mode collapse where the generation of one digit was more predominant than the rest.
- We also observed the instability in training the discriminator with the original minimax formulation of $\log(1 - G(D))$ and had to replace it with $-\log(G(D))$
- We did not find a scientific method to decide correct schedule for the number of epochs to train a discriminator for one epoch of generator or vice versa. We experimented with a few alternatives but none of them drastically altered training performance.

4.2.2 Wasserstein GAN(WGAN)

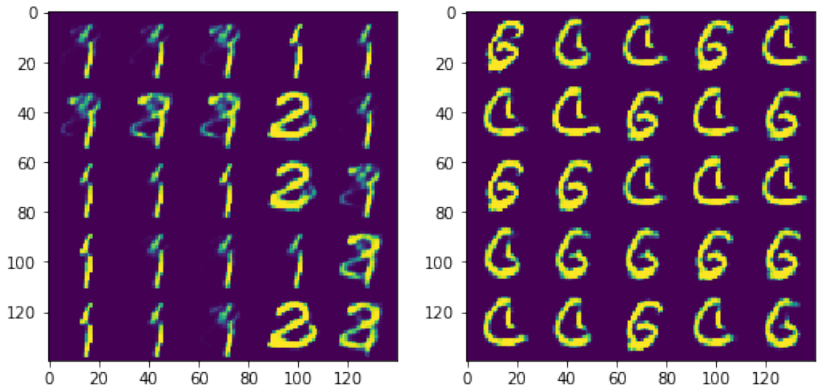


(a) Training loss (b) Vanishing gradients

Figure 11: CONFOUNDING LOSSES(WGAN) - (a) The top left corner is the loss of the generator, which is supposed to decrease. It is not clear it does on this plot. The top right corner is the loss of the discriminator, which can be broken down into to other plots (bottom left: synthetic loss, bottom right: true loss). Again, it is not clear that the discriminator improves over training, and in addition the scores of both true and synthetic samples are very similar.
(b) The left column represents the distribution of gradients for layers of the generator, and the right column those of the discriminator. We can see that most gradients of the generator are super small - which may be related to the fact that we used batch normalization.

Contrary to popular theoretical results, we did not achieve better performance while training WGANs. We tried to implement the Lipschitz constraint on the discriminator in two ways: either by the simple weight clipping, and by adding a penalty on the input gradient norm of the discriminator. In both cases, we observed poor sample quality and we also found that the loss stagnated after a few epochs as Figure 11 demonstrates.We also were plagued by mode collapse

as can be seen in Figure 12. We also note that the architecture we use (simple feed forward neural network or Deconv architectures) didn't really have an impact on the performance.



(a) epoch 12 (b) epoch 27

Figure 12: MODE COLLAPSE - As can be seen in (a) and (b), the generator in WGAN tends to produce very similar samples which don't look anything close to realistic.

Batch Norm on WGAN - When we started training WGAN, we introduced batch normalization inspired by its uses in VAE(which we didn't use for the previous GAN versions). In doing so, we observed that our discriminator did not train well. While we dont have a concrete theory why, we have a possible suspicion - The fact that the discriminator takes as inputs data coming from very different distributions (the training dataset, and the non-stationary synthetic dataset), usage of batch normalization seems incorrect - the mean and the variances are unable to converge. The consequence of this is that it prevents the discriminator to learn at all. Therefore, we removed the batch normalization from the discriminator, and reduced the learning rate.

5 Conclusion

We trained two models that learned representations of a data distribution in a continuous latent space and we could generate new samples of data via sampling z from this latent variable and running inference over it. Variational autoencoders accomplished this by introducing a family of variational parameters and GANs directly learned the latent space through adversarial training. In the process, we empirically realized certain well-known facts. GANs were able to produce sharper images than VAEs but were extremely unstable in training with observations such as non-convergence of training losses, generator mode collapses etc. Training VAEs on the other hand was less strenuous and the process was relatively more stable. Besides, as expected, using more complicated model architectures provided better results, albeit at a greater parameter tuning and computational cost. We also realized in both VAE and GAN that the generation was greatly abetted by conditioning the distributions on the class labels, helping models to learn multiple modes better.

References

Arjovsky, M., Chintala, S., and Bottou, L. (2017). Wasserstein gan. *arXiv:1701.07875*.

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. *arXiv:1406.266*.

Gulrajani, I., Kumar, K., Ahmed, F., Taiga, A. A., Visin, F., Vázquez, D., and Courville, A. C. (2016). Pixelvae: A latent variable model for natural images. *CoRR*, abs/1611.05013.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pages 448–456. JMLR.org.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

Kingma, D. P. and Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

Mirza, M. and Osindero, S. (2014). Conditional generative adversarial nets. *arXiv:1411.1784*.

Radford, A. and Metz, L. (2016). Unsupervised representation learning with deep convolutional neural networks. *arXiv:1511.06434*.

Sohn, K., Lee, H., and Yan, X. (2015). Learning structured output representation using deep conditional generative models. In Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M., and Garnett, R., editors, *Advances in Neural Information Processing Systems 28*, pages 3483–3491. Curran Associates, Inc.

van den Oord, A., Kalchbrenner, N., Espeholt, L., kavukcuoglu, k., Vinyals, O., and Graves, A. (2016). Conditional image generation with pixelcnn decoders. In Lee, D. D., Sugiyama, M., Luxburg, U. V., Guyon, I., and Garnett, R., editors, *Advances in Neural Information Processing Systems 29*, pages 4790–4798. Curran Associates, Inc.